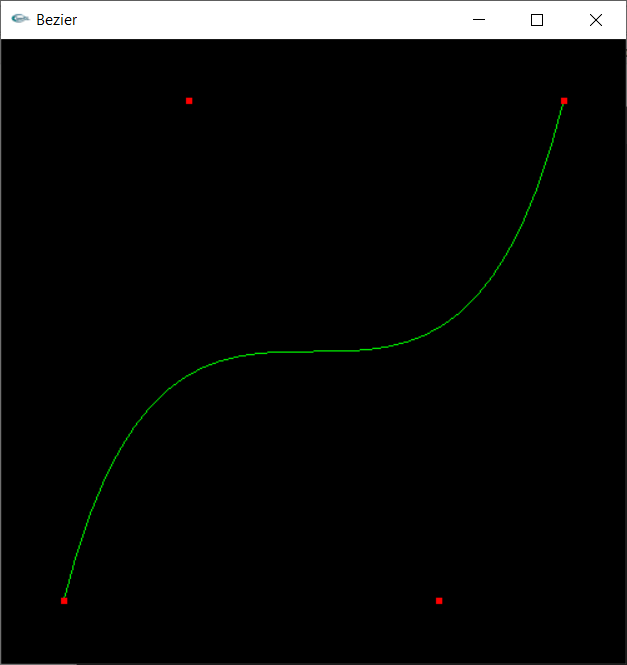
**ĐỒ HỌA MÁY TÍNH – LAB 5**

Họ và tên: Phan Trần Bảo Ngọc

Lớp: 18IT1

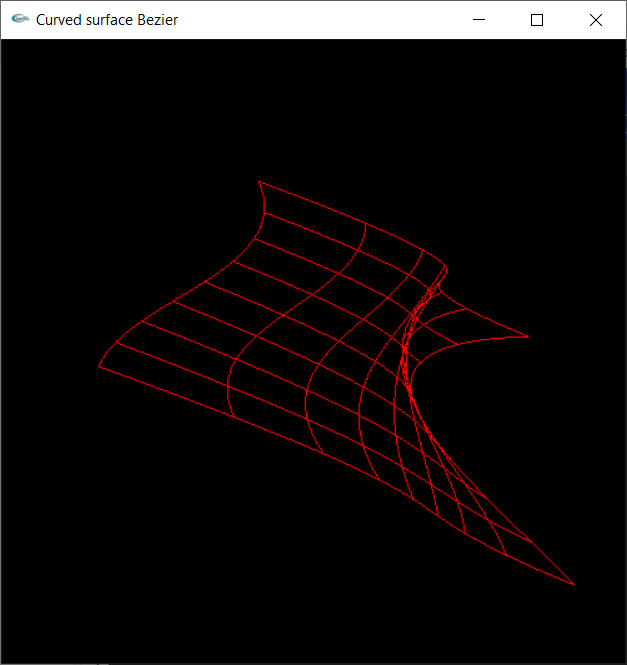
1. Đường cong Bezier

|  |
| --- |
| #include <GL/gl.h>  #include <GL/glu.h>  #include <stdlib.h>  #include <GL/glut.h>  GLfloat ctrlpoints[4][3] = {  { -4.0, -4.0, 0.0}, { -2.0, 4.0, 0.0},  {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};  void init(void)  {  glClearColor(0.0, 0.0, 0.0, 0.0);  glShadeModel(GL\_FLAT);  glMap1f(GL\_MAP1\_VERTEX\_3, 0.0, 1.0, 3, 4, &ctrlpoints[0][0]);  glEnable(GL\_MAP1\_VERTEX\_3);  }  void display(void)  {  int i;  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0, 153, 0);  glBegin(GL\_LINE\_STRIP);  for (i = 0; i <= 30; i++)  glEvalCoord1f((GLfloat) i/30.0);  glEnd();  /\* The following code displays the control points as dots. \*/  glPointSize(5.0);  glColor3f(255, 0, 0);  glBegin(GL\_POINTS);  for (i = 0; i < 4; i++)  glVertex3fv(&ctrlpoints[i][0]);  glEnd();  glFlush();  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei) w, (GLsizei) h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  if (w <= h)  glOrtho(-5.0, 5.0, -5.0\*(GLfloat)h/(GLfloat)w,  5.0\*(GLfloat)h/(GLfloat)w, -5.0, 5.0);  else  glOrtho(-5.0\*(GLfloat)w/(GLfloat)h,  5.0\*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize (500, 500);  glutInitWindowPosition (100, 100);  glutCreateWindow("Bezier");  init ();  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMainLoop();  return 0;  } |



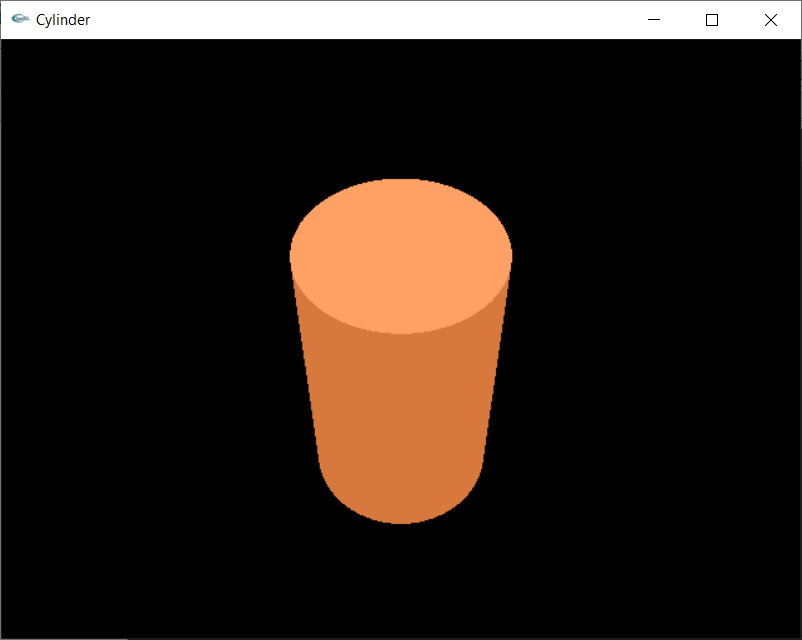
2. Mặt cong Bezier

|  |
| --- |
| #include <GL/glut.h>  #include <stdlib.h>  GLfloat ctrlpoints[4][4][3] = {  {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},  {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},  {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},  {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},  {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},  {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},  {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},  {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}  };  void display(void)  {  int i, j;  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glColor3f(1.0, 0.0, 0.0);  glPushMatrix ();  glRotatef(85.0, 1.0, 1.0, 1.0);  for (j = 0; j <= 8; j++) {  glBegin(GL\_LINE\_STRIP);  for (i = 0; i <= 30; i++)  glEvalCoord2f((GLfloat)i/30.0, (GLfloat)j/8.0);  glEnd();  glBegin(GL\_LINE\_STRIP);  for (i = 0; i <= 30; i++)  glEvalCoord2f((GLfloat)j/8.0, (GLfloat)i/30.0);  glEnd();  }  glPopMatrix ();  glFlush();  }  void init(void)  {  glClearColor (0.0,0.0,0.0, 0.0);  glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, 3, 4,  0, 1, 12, 4, &ctrlpoints[0][0][0]);  glEnable(GL\_MAP2\_VERTEX\_3);  glEnable(GL\_DEPTH\_TEST);  glShadeModel(GL\_FLAT);  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei) w, (GLsizei) h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  if (w <= h)  glOrtho(-4.0, 4.0, -4.0\*(GLfloat)h/(GLfloat)w,  4.0\*(GLfloat)h/(GLfloat)w, -4.0, 4.0);  else  glOrtho(-4.0\*(GLfloat)w/(GLfloat)h,  4.0\*(GLfloat)w/(GLfloat)h, -4.0, 4.0, -4.0, 4.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  void keyboard(unsigned char key, int x, int y)  {  switch (key) {  case 27:  exit(0);  break;  }  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);  glutInitWindowSize (500, 500);  glutInitWindowPosition (100, 100);  //glutCreateWindow (argv[0]);  glutCreateWindow("Curved surface Bezier");  init ();  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutKeyboardFunc(keyboard);  glutMainLoop();  return 0;  } |



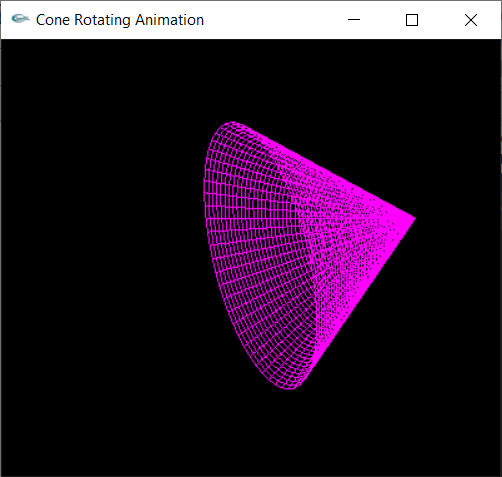
3. Cylinder

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <GL/glut.h>  #include <math.h>  #define PI 3.1415927  void draw\_cylinder(GLfloat radius,  GLfloat height,  GLubyte R,  GLubyte G,  GLubyte B)  {  GLfloat x = 0.0;  GLfloat y = 0.0;  GLfloat angle = 0.0;  GLfloat angle\_stepsize = 0.1;  glColor3ub(R-40,G-40,B-40);  glBegin(GL\_QUAD\_STRIP);  angle = 0.0;  while( angle < 2\*PI ) {  x = radius \* cos(angle);  y = radius \* sin(angle);  glVertex3f(x, y , height);  glVertex3f(x, y , 0.0);  angle = angle + angle\_stepsize;  }  glVertex3f(radius, 0.0, height);  glVertex3f(radius, 0.0, 0.0);  glEnd();  glColor3ub(R,G,B);  glBegin(GL\_POLYGON);  angle = 0.0;  while( angle < 2\*PI ) {  x = radius \* cos(angle);  y = radius \* sin(angle);  glVertex3f(x, y , height);  angle = angle + angle\_stepsize;  }  glVertex3f(radius, 0.0, height);  glEnd();  }  void display(void)  {  glClearColor (0.0, 0.0, 0.0, 0.0);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  glTranslatef(0.0,-0.4,-3.0);  glRotatef(-40, 1.0, 0.0, 0.0);  draw\_cylinder(0.3, 1.0, 255, 160, 100);  glFlush();  }  void reshape(int width, int height)  {  if (width == 0 || height == 0) return;  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(40.0, (GLdouble)width/(GLdouble)height,  0.5, 20.0);  glMatrixMode(GL\_MODELVIEW);  glViewport(0, 0, width, height);  }  int main(int argc, char \*\*argv)  {    glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize(640,480);  glutCreateWindow("Cylinder");  glClearColor(1.0,1.0,1.0,0.0);  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMainLoop();  return 0;  } |



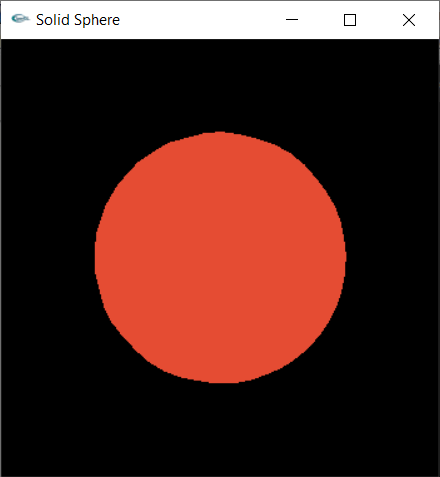
4. Cone

|  |
| --- |
| #include <GL\glut.h>  GLfloat xRotated, yRotated, zRotated;  // Cone  GLdouble base=1;  GLdouble height=1.5;  GLint slices =50;  GLint stacks =50;  void displayCone(void)  {  glMatrixMode(GL\_MODELVIEW);  // clear the drawing buffer.  glClear(GL\_COLOR\_BUFFER\_BIT);  // clear the identity matrix.  glLoadIdentity();  // traslate the draw by z = -4.0  // Note this when you decrease z like -8.0 the drawing will looks far , or smaller.  glTranslatef(0.0,0.0,-4.5);    glColor3f(204, 0, 153);  // changing in transformation matrix.  // rotation about X axis  glRotatef(xRotated,1.0,0.0,0.0);  // rotation about Y axis  glRotatef(yRotated,0.0,1.0,0.0);  // rotation about Z axis  glRotatef(zRotated,0.0,0.0,1.0);  // scaling transfomation  glScalef(1.0,1.0,1.0);  // built-in (glut library) function , draw you a Cone.  glutSolidCone(base,height,slices,stacks);  // Flush buffers to screen  glFlush();  // sawp buffers called because we are using double buffering  // glutSwapBuffers();  }  void reshapeCone(int x, int y)  {  if (y == 0 || x == 0) return; //Nothing is visible then, so return  //Set a new projection matrix  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  //Angle of view:40 degrees  //Near clipping plane distance: 0.5  //Far clipping plane distance: 20.0  gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);    glViewport(0,0,x,y); //Use the whole window for rendering  }  void idleCone(void)  {  yRotated += 0.01;    displayCone();  }  int main (int argc, char \*\*argv)  {  //Initialize GLUT  glutInit(&argc, argv);  //double buffering used to avoid flickering problem in animation  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  // window size  glutInitWindowSize(400,350);  // create the window  glutCreateWindow("Cone Rotating Animation");  glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_LINE);  xRotated = yRotated = zRotated = 30.0;  xRotated=33;  yRotated=40;  glClearColor(0.0,0.0,0.0,0.0);  //Assign the function used in events  glutDisplayFunc(displayCone);  glutReshapeFunc(reshapeCone);  glutIdleFunc(idleCone);  //Let start glut loop  glutMainLoop();  return 0;  } |



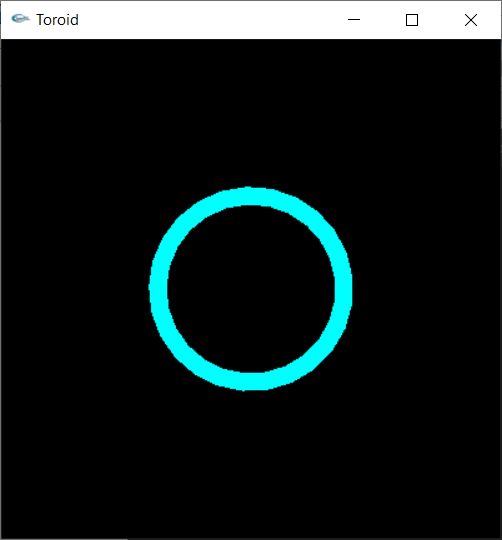
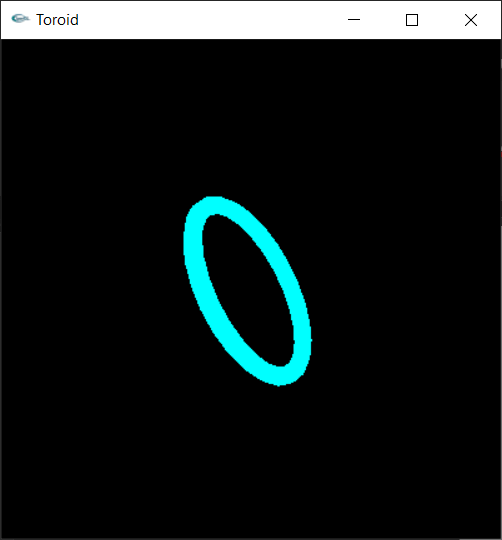
5. Sphere

|  |
| --- |
| #include <GL\glut.h>  GLfloat xRotated, yRotated, zRotated;  GLdouble radius=1;  void display(void);  void reshape(int x, int y);  int main (int argc, char \*\*argv)  {  glutInit(&argc, argv);  glutInitWindowSize(350,350);  glutCreateWindow("Solid Sphere");  xRotated = yRotated = zRotated = 30.0;  xRotated=43;  yRotated=50;    glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMainLoop();  return 0;  }  void display(void)  {  glMatrixMode(GL\_MODELVIEW);  // clear the drawing buffer.  glClear(GL\_COLOR\_BUFFER\_BIT);  // clear the identity matrix.  glLoadIdentity();  // traslate the draw by z = -4.0  // Note this when you decrease z like -8.0 the drawing will looks far , or smaller.  glTranslatef(0.0,0.0,-5.0);  // Red color used to draw.  glColor3f(0.9, 0.3, 0.2);  // changing in transformation matrix.  // rotation about X axis  glRotatef(xRotated,1.0,0.0,0.0);  // rotation about Y axis  glRotatef(yRotated,0.0,1.0,0.0);  // rotation about Z axis  glRotatef(zRotated,0.0,0.0,1.0);  // scaling transfomation  glScalef(1.0,1.0,1.0);  // built-in (glut library) function , draw you a sphere.  glutSolidSphere(radius,20,20);  // Flush buffers to screen    glFlush();  // sawp buffers called because we are using double buffering  // glutSwapBuffers();  }  void reshape(int x, int y)  {  if (y == 0 || x == 0) return;  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(39.0,(GLdouble)x/(GLdouble)y,0.6,21.0);  glMatrixMode(GL\_MODELVIEW);  glViewport(0,0,x,y); //Use the whole window for rendering  } |

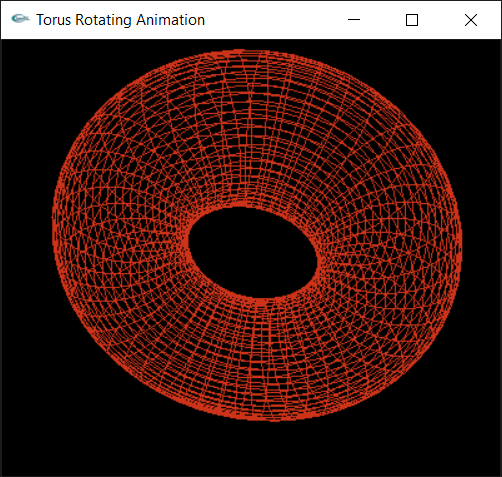


6.

|  |
| --- |
| #include <GL/glut.h>  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #define PI\_ 3.14159265358979323846  GLuint theTorus;  /\* Draw a torus \*/  static void torus(int numc, int numt)  {  int i, j, k;  double s, t, x, y, z, twopi;  twopi = 2 \* PI\_;  for (i = 0; i < numc; i++) {  glBegin(GL\_QUAD\_STRIP);  for (j = 0; j <= numt; j++) {  for (k = 1; k >= 0; k--) {  s = (i + k) % numc + 0.5;  t = j % numt;  x = (1+.1\*cos(s\*twopi/numc))\*cos(t\*twopi/numt);  y = (1+.1\*cos(s\*twopi/numc))\*sin(t\*twopi/numt);  z = .1 \* sin(s \* twopi / numc);  glVertex3f(x, y, z);  }  }  glEnd();  }  }  /\* Create display list with Torus and initialize state \*/  static void init(void)  {  theTorus = glGenLists (1);  glNewList(theTorus, GL\_COMPILE);  torus(8, 25);  glEndList();  glShadeModel(GL\_FLAT);  glClearColor(0.0, 0.0, 0.0, 0.0);  }  /\* Clear window and draw torus \*/  void display(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f (0, 153, 153);  glCallList(theTorus);  glFlush();  }  /\* Handle window resize \*/  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei) w, (GLsizei) h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(30, (GLfloat) w/(GLfloat) h, 1.0, 100.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);  }  void keyboard(unsigned char key, int x, int y)  {  switch (key) {  case 'x':  case 'X':  glRotatef(30.,1.0,0.0,0.0);  glutPostRedisplay();  break;  case 'y':  case 'Y':  glRotatef(30.,0.0,1.0,0.0);  glutPostRedisplay();  break;  case 'i':  case 'I':  glLoadIdentity();  gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);  glutPostRedisplay();  break;  case 27:  exit(0);  break;  }  }  int main(int argc, char \*\*argv)  {  glutInitWindowSize(400, 400);  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutCreateWindow("Toroid");  init();  glutReshapeFunc(reshape);  glutKeyboardFunc(keyboard);  glutDisplayFunc(display);  glutMainLoop();  return 0;  } |

|  |
| --- |
| #include <GL\glut.h>  GLfloat xRotated, yRotated, zRotated;  // Torus  GLdouble innerRaidus=0.5;  GLdouble outterRaidus=1;  GLint sides =50;  GLint rings =50;  void displayTorus(void)  {  glMatrixMode(GL\_MODELVIEW);  // clear the drawing buffer.  glClear(GL\_COLOR\_BUFFER\_BIT);  // clear the identity matrix.  glLoadIdentity();  // traslate the draw by z = -4.0  // Note this when you decrease z like -8.0 the drawing will looks far , or smaller.  glTranslatef(0.0,0.0,-4.5);  // Red color used to draw.  glColor3f(0.8, 0.2, 0.1);  // changing in transformation matrix.  // rotation about X axis  glRotatef(xRotated,1.0,0.0,0.0);  // rotation about Y axis  glRotatef(yRotated,0.0,1.0,0.0);  // rotation about Z axis  glRotatef(zRotated,0.0,0.0,1.0);  // scaling transfomation  glScalef(1.0,1.0,1.0);  // built-in (glut library) function , draw you a Torus.    glutSolidTorus(innerRaidus,outterRaidus,sides,rings);  // Flush buffers to screen    glFlush();  // sawp buffers called because we are using double buffering  // glutSwapBuffers();  }  void reshapeTorus(int x, int y)  {  if (y == 0 || x == 0) return; //Nothing is visible then, so return  //Set a new projection matrix  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  //Angle of view:40 degrees  //Near clipping plane distance: 0.5  //Far clipping plane distance: 20.0    gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);    glViewport(0,0,x,y); //Use the whole window for rendering  }  void idleTorus(void)  {    yRotated += 0.01;    displayTorus();  }  int main (int argc, char \*\*argv)  {  //Initialize GLUT  glutInit(&argc, argv);  //double buffering used to avoid flickering problem in animation  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  // window size  glutInitWindowSize(400,350);  // create the window  glutCreateWindow("Torus Rotating Animation");  glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_LINE);  xRotated = yRotated = zRotated = 30.0;  xRotated=33;  yRotated=40;  glClearColor(0.0,0.0,0.0,0.0);  //Assign the function used in events  glutDisplayFunc(displayTorus);  glutReshapeFunc(reshapeTorus);  glutIdleFunc(idleTorus);  //Let start glut loop  glutMainLoop();  return 0;  } |



6. Hyperboloid

|  |
| --- |
| #include <cmath>  #include <iostream>  # include <GL/glut.h>  #define PI 3.14159265358979324  using namespace std;  // Globals.  static int p = 8; // Number of grid columns.  static int q = 8; // Number of grid rows  static float \*vertices = NULL; // Vertex array of the mapped sample on the surface.  static float Xangle = 330.0, Yangle = 0.0, Zangle = 0.0; // Angles to rotate the surface.  // Fuctions to map the grid vertex (u\_i,v\_j) to the mesh vertex (f(u\_i,v\_j), g(u\_i,v\_j), h(u\_i,v\_j)) on the patch.  float f(int i, int j)  {  return ( cos((-1+2.0\*(float)i/p)\*PI) / cos((-0.4+0.8\*(float)j/q)\*PI) );  }  float g(int i, int j)  {  return ( sin((-1+2.0\*(float)i/p)\*PI) / cos((-0.4+0.8\*(float)j/q)\*PI) );  }  float h(int i, int j)  {  return ( tan((-0.4+0.8\*(float)j/q)\*PI) );  }  // Routine to fill the vertex array with co-ordinates of the mapped sample points.  void fillVertexArray(void)  {  int i, j, k;  k = 0;  for (j = 0; j <= q; j++)  for (i = 0; i <= p; i++)  {  vertices[k++] = f(i,j);  vertices[k++] = g(i,j);  vertices[k++] = h(i,j);  }  }  // Initialization routine.  void setup(void)  {  glEnableClientState(GL\_VERTEX\_ARRAY);    glClearColor(0,0,0,0);  }  // Drawing routine.  void drawScene(void)  {  int i, j;  vertices = new float[3\*(p+1)\*(q+1)]; // Dynamic array allocation with new value of p and q.  glVertexPointer(3, GL\_FLOAT, 0, vertices);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  gluLookAt (0.0, 0.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);  glColor3f(0, 153, 153);    // Rotate scene.  glRotatef(Zangle, 0.0, 0.0, 1.0);  glRotatef(Yangle, 0.0, 1.0, 0.0);  glRotatef(Xangle, 1.0, 0.0, 0.0);  // Fill the vertex array.  fillVertexArray();  // Make the approximating triangular mesh.  for(j = 0; j < q; j++)  {  glBegin(GL\_TRIANGLE\_STRIP);  for(i = 0; i <= p; i++)  {  glArrayElement( (j+1)\*(p+1) + i );  glArrayElement( j\*(p+1) + i );  }  glEnd();  }  glutSwapBuffers();  }  // OpenGL window reshape routine.  void resize(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(60.0, (float)w/(float)h, 1.0, 50.0);  glMatrixMode(GL\_MODELVIEW);  }  // Keyboard input processing routine.  void keyInput(unsigned char key, int x, int y)  {  switch(key)  {  case 27:  exit(0);  break;  case 'x':  Xangle += 5.0;  if (Xangle > 360.0) Xangle -= 360.0;  glutPostRedisplay();  break;  case 'X':  Xangle -= 5.0;  if (Xangle < 0.0) Xangle += 360.0;  glutPostRedisplay();  break;  case 'y':  Yangle += 5.0;  if (Yangle > 360.0) Yangle -= 360.0;  glutPostRedisplay();  break;  case 'Y':  Yangle -= 5.0;  if (Yangle < 0.0) Yangle += 360.0;  glutPostRedisplay();  break;  case 'z':  Zangle += 5.0;  if (Zangle > 360.0) Zangle -= 360.0;  glutPostRedisplay();  break;  case 'Z':  Zangle -= 5.0;  if (Zangle < 0.0) Zangle += 360.0;  glutPostRedisplay();  break;  default:  break;  }  }  // Callback routine for non-ASCII key entry.  void specialKeyInput(int key, int x, int y)  {  if (key == GLUT\_KEY\_LEFT) if (p > 3) p -= 1;  if (key == GLUT\_KEY\_RIGHT) p += 1;  if (key == GLUT\_KEY\_DOWN) if (q > 3) q -= 1;  if (key == GLUT\_KEY\_UP) q += 1;  glutPostRedisplay();  }  // Routine to output interaction instructions to the C++ window.  void printInteraction(void)  {  cout << "Interaction:" << endl;  cout << "Press left/right arrow keys to increase/decrease the number of grid columns." << endl  << "Press up/down arrow keys to increase/decrease the number of grid rows." << endl  << "Press x, X, y, Y, z, Z to turn the surface." << endl;  }  // Main routine.  int main(int argc, char \*\*argv)  {  printInteraction();  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);  glutInitWindowSize(500, 500);  glutInitWindowPosition(100, 100);  glutCreateWindow("Hyperboloid");  setup();  glutDisplayFunc(drawScene);  glutReshapeFunc(resize);  glutKeyboardFunc(keyInput);  glutSpecialFunc(specialKeyInput);  glutMainLoop();  return 0;  } |

